1ST MAKER SPACE
Design.
Build.
Sustain.

Frog Microcontroller Trainer (FMT): Stopwatch

Length of Time: 2 days x 55 Minutes

Recommended for Advanced

Contents

Overview:

This lesson guides high school students through building a fully functional stopwatch using an Arduino-based 1ST Maker Frog board. Students apply prior knowledge of buttons, displays, and timing to write a program from scratch that tracks elapsed time, supports start/stop cycles, records a lap, and displays time in a formatted M:SS.T layout

Objectives:

Students will be able to:

Compute elapsed time using millis() by saving a start timestamp and subtracting it each loop.

Implement a start/stop pattern that correctly accumulates time across multiple run/stop cycles using a banked-time variable (accumulatedMs).

Save a single lap time and display it alongside the running elapsed time.

Format a millisecond integer as M:SS.T (minutes, seconds, tenths) without floating-point math.

Apply the button debounce and edge-detection patterns from prior programs to a new project.

Write a complete, working Arduino program from a specification, using prior curriculum code as reference.

Computer Science Teacher Association Standards:

- 3A-CS-03 – Develop guidelines that convey systematic troubleshooting strategies that others can use to identify and fix errors.
- 3A-AP-13 – Create prototypes that use algorithms to solve computational problems by leveraging prior student knowledge and personal interests.
- 3A-AP-15 – Justify the selection of specific control structures in terms of readability and performance.
- 3A-AP-16 – Design and iteratively develop computational artifacts for practical intent, personal expression, or to address a societal issue by using events to initiate instructions.
- 3A-AP-17 – Decompose problems into smaller components through systematic analysis, using constructs such as procedures, modules, and/or objects.
- 3A-AP-21 – Evaluate and refine computational artifacts to make them more usable and accessible.
- 3A-AP-23 – Document design decisions using text, graphics, presentations, and/or demonstrations in the development of complex programs.

Materials:

- Frog Microcontroller Trainer from 1st Maker Space
- USB Power Cord
- Arduino IDE
- PC or Mac
- Chromebooks if using Arduino Create for Education App
- Copies of Student Handout

Preparation:

- Run the program yourself to see how it works.
- Read the ‘teacher troubleshooting notes’.
- Print and distribute the student handouts.

Background Information:

Why millis() Instead of delay():

Every program in this curriculum that runs smoothly over time — Countdown, Chess Timer, Button Battle — uses millis() rather than delay() to measure time. delay() blocks the entire program: no buttons can be read, no display can be updated, and nothing else can happen until the delay finishes. millis() returns the number of milliseconds since the board powered on and never blocks. Programs that use millis() run through loop() as fast as possible and simply check whether enough time has passed to take an action.

Computing Elapsed Time:

A stopwatch needs to report how much total time has passed. The technique is:

elapsed = millis() - startMs;
where startMs is the value of millis() captured at the moment the timer started. As long as the timer is running, this expression gives a continuously increasing elapsed time in milliseconds.

Banking Time Across Start/Stop Cycles:

A real stopwatch can be stopped and restarted without resetting. This requires saving ("banking") the elapsed time before each stop, then adding new time on the next run. Two variables handle this:

- accumulatedMs — total ms banked from all completed run intervals.
- startMs — millis() captured when the current run began.

The elapsed time at any moment is:

```
elapsed = accumulatedMs + (millis() - startMs);   // while running
elapsed = accumulatedMs;                          // while stopped
```

When the stop button is pressed, the current elapsed time is calculated and saved into accumulatedMs. When the start button is pressed again, startMs is refreshed to the current millis(). The next call to getElapsedMs() adds the new run's time on top of what was banked.

Recording and Displaying a Lap:

Recording the current time as a lap is one line: save the result of getElapsedMs() into a variable. A boolean flag (hasLap) tracks whether a lap has been saved so the display knows whether to show it. This is simpler than an array and sufficient for one lap.

```
lastLapMs = getElapsedMs();
hasLap    = true;
```

Displaying Tenths of a Second:

Dividing a millisecond value by 1000 gives whole seconds but discards the sub-second part. To display tenths, divide by 100 first to get total tenths, then extract each digit with modulo and integer division — no floating-point needed:

```
uint32_t t = ms / 100;          // total tenths
uint8_t  tenths  = t % 10;
uint8_t  seconds = (t / 10) % 60;
uint32_t minutes = t / 600;     // 600 tenths = 1 minute
```

snprintf() then assembles the string. The format specifier %02u pads seconds with a leading zero so "0:05.3" appears instead of "0:5.3".

Combined Button Detector:

This program uses the same debounce and rising-edge pattern from Button Battle and Chess Timer. Key points:

- A time-based debounce filter only updates the stable state after the raw signal has been consistent for at least DEBOUNCE_MS milliseconds, filtering out contact bounce.
- A one-shot press event (leftPress, rightPress) is set true for exactly one loop iteration on the rising edge, so holding a button does not repeat the action.
- Check the both-buttons case first and return early so pressing both never also fires a single-button action on the same loop.

Program Architecture Reference:

This section describes the solution architecture. Use it to guide stuck students or as a pre-challenge discussion. Do not show High_Low.ino to students until after they have made a genuine attempt.

State Machine:

State	Meaning	Transition To
STATE_STOPPED	Timer is paused. accumulatedMs holds the banked time.	STATE_RUNNING when LEFT is pressed.
STATE_RUNNING	Timer is counting. Elapsed = accumulatedMs + (millis() − startMs).	STATE_STOPPED when LEFT is pressed. Also STOPPED on reset.

Key Variables:

Variable	Type	Purpose
state	SwState (enum)	Current FSM state. Checked in handleButtons() and all output functions.
startMs	uint32_t	millis() captured when the timer last started. Used only while running.
accumulatedMs	uint32_t	Total ms banked from previous runs. Saved on each Stop press.
lastLapMs	uint32_t	Elapsed ms at the moment the lap button was pressed.
hasLap	bool	True once a lap has been saved. Controls the lap line on the display and LED[0].
leftStable / rightStable	bool	Debounced button states. true = currently pressed.
leftPress / rightPress	bool	One-shot press events. true for one loop iteration only.

Functions:

Function	Responsibility
initHardware()	Set pin modes, start OLED, initialize NeoPixels.
updateButtons(now)	Debounce both buttons; set leftPress / rightPress on rising edges.
handleButtons(now)	Act on events: both=reset, left=start/stop, right=save/clear lap.
getElapsedMs()	Return accumulatedMs + (millis() − startMs) if running, else accumulatedMs.
updateDisplay()	Clear display, draw status, elapsed time, and lap line.
formatTime(ms, buf, sz)	Convert ms to M:SS.T string using integer math and snprintf().
updateEyes()	Green NeoPixels when running; off when stopped.

Teacher Troubleshooting Notes:

► TEACHER: Timer resets to zero on every Stop: the student is calling reset logic inside the stop branch. Have them trace handleButtons() and identify which branch runs on the first left press.

► TEACHER: Elapsed time runs at double speed: startMs is being reset to millis() every loop rather than only on the press event. Confirm that startMs = now is inside the if (leftPress ...) block, not in the else (STATE_STOPPED) branch unconditionally.

► TEACHER: Display flickers badly: the student is calling display.display() every loop. All display calls must be inside the if (now − lastDisplayMs >= DISPLAY_MS) block.

► TEACHER: No OLED output at all: ask the student to add Serial.begin(9600) and Serial.println("setup done") to verify setup() is running. If it is, the issue is in initHardware() or the I2C address.

► TEACHER: Lap time shows 0:00.0: the student is saving lastLapMs before getElapsedMs() is implemented, or their state variable is STATE_STOPPED at the moment recordLap fires. Add a Serial.println(getElapsedMs()) inside the lap branch to confirm the value being saved.

► TEACHER: tone() followed immediately by a second tone() plays neither: the hardware timer is restarted before the first tone completes. Keep all tone() calls at least 50 ms apart in real time, or just play one tone per button event.

Lesson Elements	
Introduction 10-15 minutes	Demo the finished stopwatch (run the teacher solution). Show start/stop, save a lap, reset. Ask: What state is it in now? What happens if I press SW1?
Concept – 20 minutes	Teach sections elapsed time and banking. Draw the formula accumulatedMs + (millis() − startMs) on the board and trace through a start → stop → start → stop cycle by hand.
Challenge – 50-70 minutes	Students build the program following the step-by-step. Circulate and use the troubleshooting guide. Note: See a potential solution for the program at the end of this document.
Debrief– 10 minutes	Ask: What is the difference between startMs and accumulatedMs? What happens if you forget to save accumulatedMs on Stop? Why does the display only refresh every 50 ms instead of every loop?
Career Exploration: • A good resource is the IDOE Career Explorer database. • Another good resource for career information is the Bureau of Labor Statistics	
Practical Application: The stopwatch you are building is not just a classroom exercise—it models how real devices and systems track time, respond to user input, and manage ongoing processes.	

Additional Resources:

- 1st Maker Space Frog Microcontroller Library
- 1st Maker Space Learn Arduino with the Frog Microcontroller

Performance Assessment/Check for Understanding :

- Was the student able to successfully write the program for the FMT?
- Check student handout for responses.

Additional Feedback:

Potential Solution:

```
/************************************************
 * Program: Stopwatch.ino
 * Copyright (C) 1st Maker Space, Inc. 2026
 *
 * Description:
 *   A simple stopwatch with single-lap memory.
 *
 *   LEFT  button : Start or Stop the timer.
 *   RIGHT button : Save the current time as a lap
 *                  (while running), or clear the
 *                  saved lap (while stopped).
 *   BOTH  buttons: Reset everything to zero.
 *
 *   Elapsed time and last lap are shown on the OLED
 *   in M:SS.T format. NeoPixel eyes glow green while
 *   running.
 *
 * Hardware:
 *   - 1ST Maker Frog board (Arduino UNO R4 Minima)
 *   - OLED display (SSD1306, 128x64) via I2C
 *   - LEFT  button = BUTTON_ONE (pin 1) [active-LOW]
 *   - RIGHT button = BUTTON_TWO (pin 0) [active-LOW]
 *   - NeoPixel eyes, piezo buzzer
 *
 * Teacher note:
 *   This is the proposed SOLUTION. Do not distribute
 *   to students before they attempt the challenge.
 ************************************************/

#include "1ST_Maker_Frog.h"

// ---------------------------------------------------------------------
//  Configuration
// ---------------------------------------------------------------------

const uint16_t DEBOUNCE_MS = 30;    // button debounce hold time in ms
const uint16_t DISPLAY_MS  = 50;    // OLED refresh interval in ms (~20 fps)

// ---------------------------------------------------------------------
//  State machine
// ---------------------------------------------------------------------

enum SwState { STATE_STOPPED, STATE_RUNNING };
SwState state = STATE_STOPPED;
```

```
// ---------------------------------------------------------------------------
//  Timer and lap variables
// ---------------------------------------------------------------------------

uint32_t startMs       = 0;     // millis() when the timer last started
uint32_t accumulatedMs = 0;     // time banked from all previous runs
uint32_t lastLapMs     = 0;     // elapsed ms when lap was recorded
bool     hasLap        = false; // true once a lap has been saved

uint32_t lastDisplayMs = 0;

// ---------------------------------------------------------------------------
//  Button debounce variables
// ---------------------------------------------------------------------------

bool     leftStable    = false;
bool     rightStable   = false;
uint32_t leftChangeMs  = 0;
uint32_t rightChangeMs = 0;
bool     leftPress     = false;  // one-shot: true for one loop when pressed
bool     rightPress    = false;

// ---------------------------------------------------------------------------
//  Function prototypes
// ---------------------------------------------------------------------------

void     initHardware();
void     updateButtons(uint32_t now);
void     handleButtons(uint32_t now);
uint32_t getElapsedMs();
void     updateDisplay();
void     formatTime(uint32_t ms, char* out, size_t outSize);
void     updateEyes();

// ---------------------------------------------------------------------------
//  Setup and loop
// ---------------------------------------------------------------------------

void setup() {
  initHardware();
}

void loop() {
```

```
  uint32_t now = millis();

  updateButtons(now);
  handleButtons(now);

  if (now - lastDisplayMs >= DISPLAY_MS) {
    lastDisplayMs = now;
    updateDisplay();
  }

  updateEyes();
}

//
=================================================================
=========
//  Functions
//
=================================================================
=========

void initHardware() {
  pinMode(BUTTON_ONE, INPUT_PULLUP);
  pinMode(BUTTON_TWO, INPUT_PULLUP);
  pinMode(PIEZO, OUTPUT);
  pixel.begin();
  pixel.show();
  Wire.begin();
  display.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS);
  display.clearDisplay();
  display.display();
}

/***********************************************
 * Function: updateButtons
 * Purpose:  Debounce both buttons and set one-shot
 *           press events for this loop iteration.
 *           leftPress / rightPress are true for
 *           exactly one loop on the rising edge
 *           (not-pressed → pressed transition).
 ***********************************************/
void updateButtons(uint32_t now) {
  leftPress  = false;
  rightPress = false;
```

```
  bool rawLeft  = (digitalRead(BUTTON_ONE) == PRESSED);
  bool rawRight = (digitalRead(BUTTON_TWO) == PRESSED);

  // Left button
  if (rawLeft != leftStable) {
    if (now - leftChangeMs >= DEBOUNCE_MS) {
      bool prev  = leftStable;
      leftStable = rawLeft;
      leftChangeMs = now;
      if (leftStable && !prev) leftPress = true;  // rising edge
    }
  } else {
    leftChangeMs = now;
  }

  // Right button (same logic)
  if (rawRight != rightStable) {
    if (now - rightChangeMs >= DEBOUNCE_MS) {
      bool prev   = rightStable;
      rightStable = rawRight;
      rightChangeMs = now;
      if (rightStable && !prev) rightPress = true;
    }
  } else {
    rightChangeMs = now;
  }
}

/*************************************************
 * Function: handleButtons
 * Purpose:  Act on button press events.
 *
 *   BOTH held  → reset everything to zero.
 *   LEFT alone → start or stop the timer.
 *   RIGHT alone→ save a lap (running) or
 *                clear the saved lap (stopped).
 *
 *   The !rightStable / !leftStable guards prevent
 *   a two-button reset from also firing a single-
 *   button action on the same loop iteration.
 *************************************************/
void handleButtons(uint32_t now) {
  // BOTH held → reset
```

```
  if (leftStable && rightStable) {
    state        = STATE_STOPPED;
    accumulatedMs = 0;
    startMs       = 0;
    lastLapMs     = 0;
    hasLap        = false;
    tone(PIEZO, 600, 150);
    return;
  }

  // LEFT alone → start or stop
  if (leftPress && !rightStable) {
    if (state == STATE_RUNNING) {
      accumulatedMs = getElapsedMs();  // bank elapsed time before stopping
      state = STATE_STOPPED;
    } else {
      startMs = now;                   // record start of this run
      state   = STATE_RUNNING;
    }
    tone(PIEZO, 1500, 40);
  }

  // RIGHT alone → save lap or clear lap
  if (rightPress && !leftStable) {
    if (state == STATE_RUNNING) {
      lastLapMs = getElapsedMs();
      hasLap    = true;
    } else {
      hasLap = false;
    }
    tone(PIEZO, 1500, 40);
  }
}
/*************************************************
 * Function: getElapsedMs
 * Purpose:  Return total elapsed time in ms.
 *
 *   Running: banked time + time since last start.
 *   Stopped: banked time only.
 *
 * This is the core of the stopwatch. Adding
 * accumulatedMs to (millis() - startMs) combines
 * all previous run intervals with the current one.
 *************************************************/
```

```
uint32_t getElapsedMs() {
  if (state == STATE_RUNNING) {
    return accumulatedMs + (millis() - startMs);
  }
  return accumulatedMs;
}

/***********************************************
 * Function: updateDisplay
 * Purpose:  Draw elapsed time and lap on the OLED.
 *
 * Layout:
 *   Top (small):    "RUNNING" or "STOPPED"
 *   Middle (large): elapsed time in M:SS.T
 *   Bottom (small): "LAP: M:SS.T" or a hint
 ***********************************************/
void updateDisplay() {
  display.clearDisplay();
  display.setTextColor(SSD1306_WHITE);

  // Status line
  display.setTextSize(1);
  display.setCursor(0, 0);
  display.print(state == STATE_RUNNING ? "RUNNING" : "STOPPED");

  // Elapsed time — large, horizontally centered
  char buf[10];
  formatTime(getElapsedMs(), buf, sizeof(buf));
  display.setTextSize(3);
  int16_t x = (SCREEN_WIDTH - (int16_t)(strlen(buf) * 18)) / 2;
  display.setCursor(max(x, (int16_t)0), 14);
  display.print(buf);

  // Lap line or hint
  display.setTextSize(1);
  display.setCursor(0, 52);
  if (hasLap) {
    char lapBuf[10];
    formatTime(lastLapMs, lapBuf, sizeof(lapBuf));
    display.print("LAP: ");
    display.print(lapBuf);
  } else {
    display.print(state == STATE_RUNNING ? "SW2: lap" : "SW1: start");
  }
```

```
  display.display();
}

/************************************************
 * Function: formatTime
 * Purpose:  Format ms as "M:SS.T" using integer math.
 *
 * Example: 65340 ms → "1:05.3"
 *
 * Divide by 100 to get total tenths, then use
 * modulo and division to pull out each digit.
 * No floating-point math needed.
 ************************************************/
void formatTime(uint32_t ms, char* out, size_t outSize) {
  uint32_t t = ms / 100;           // total tenths of a second
  uint8_t  tenths  = t % 10;
  uint8_t  seconds = (t / 10) % 60;
  uint32_t minutes = t / 600;      // 600 tenths = 1 minute
  snprintf(out, outSize, "%lu:%02u.%u", minutes, seconds, tenths);
}

/************************************************
 * Function: updateEyes
 * Purpose:  Green eyes while running, off when stopped.
 ************************************************/
void updateEyes() {
  if (state == STATE_RUNNING) {
    pixel.setPixelColor(0, pixel.Color(0, 60, 0));
    pixel.setPixelColor(1, pixel.Color(0, 60, 0));
  } else {
    pixel.setPixelColor(0, 0);
    pixel.setPixelColor(1, 0);
  }
  pixel.show();
}
```